VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace knihovny základních funkcí a sady příkladů použití NetLink socket pro řízení Linuxového jádra

Implementation of a Simple Library for a Linux Kernel configuration Based on the NetLink Socket

2013 Pavel Kovář

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Kovář**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace knihovny základních funkcí a sady příkladů použití NetLink socket pro řízení Linuxového jádra
Implementation of a Simple Library for a Linux Kernel configuration Based on the NetLink Socket

Zásady pro vypracování:

Cílem práce je zefektivnit použití NetLink socket v jazyce Python při provádění základních operací souvisejících s konfigurací sítě a síťových služeb v prostředí Linux/GNU a dále připravit sadu příkladů použití NetLink ve specifických případech konfigurace. Knihovní balíček bude obsahovat zejména prvky potřebné pro nastavování IP adres na rozhraní a řízení směrování.

1. Seznamte se s přístupem konfigurace operačního systému Linux/GNU prostřednictvím NetLink soketů.
2. Prostudujte a ověřte na praktické implementaci zprávy určené ke konfiguraci síťového subsystému a to zejména konfigurace adres síťových rozhraní (Ethernet a IP adresy včetně doplňkových informací) a konfigurace lokálního směrování v hlavní směrovací tabulce systému.
3. Implementujte balíček, který zefektivní práci se síťovým subsystémam v jazyce Python.
4. Výsledky řádně otestujte na sadě příkladů použití vytvořeného balíčku na ukázkách, které budou realizovat manipulaci s adresami síťového rozhraní a modifikaci směrovací tabulky.
5. Zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] SMITH, Roderick W. Advanced Linux networking. USA: Addison-Wesley, 2002, 752 s. ISBN 9780201774238
[2] DONG, Jielin. Network Protocols Handbook 4th edition. USA: Javvin Technologies Inc., 2005, 380 s. ISBN 978-1-60267-002-0

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Milata**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. 5 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. 5 2013 .....

Rád bych na tomto místě poděkoval ing. Martinu Milatovi, za odborné vedení a přínosnou kritiku mé práce.

Abstrakt

Cílem práce je zefektivnit použití NetLink socket v jazyce Python při provádění základních operací souvisejících s konfigurací sítě a síťových služeb v prostředí Linux/GNU a dále připravit sadu příkladů použití NetLink ve specifických případech konfigurace. Knihovní balíček bude obsahovat zejména prvky potřebné pro nastavování IP adres na rozhraní a řízení směrování.

Klíčová slova: Linux, Netlink, Python, Balíček

Abstract

The goal of this bachelor work is icrease efectivity of basic configuration tasks of network and network services trough Netlink socket in Linux/GNU environment. Then prepare a set of examples of using Netlink in specific cases of configurations. The library package will contain especialy tools nesecary for configuring IP addresses on interfaces and for routing control.

Keywords: Linux, Netlink, Python, Package

Seznam použitých zkratek a symbolů

IP	–	Internet Protocol
IPv4	–	Internet Protocol verze 4
IPv6	–	Internet Protocol verze 6

Obsah

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

1.1 Co je to Netlink socket

Netlink socket je obecně mechanizmus meziprocesové komunikace v operačním systému Linux. Je ho možné použít jak pro výměnu informací mezi procesy bežícími v uživatelském prostoru, tak i tehdy, když některý z procesů běží v jádře. Současná podoba tohoto mechanismu je dostupná od jádra verze 2.2.

Tato práce se zabývá využitím tohoto mechanizmu pro konfiguraci síťového subsystému v jádře. K tomuto účelu bylo v jádře vytvořeno několik služeb, se kterými se komunikuje právě přes Netlink socket. Tyto služby vznikly jako snaha sjednotit a usnadnit tuto konfiguraci.

V této práci se zaměřím zejména na konfiguraci adres na síťových rozhraních a na modifikaci směrovacích tabulek.

1.2 Motivace

Hlavním důvodem pro vznik této práce je fakt, že v podstatě jedinou dostupnou dokumentací je tento dokument [2], který ale popisuje pouze směrování a RFC3549 [3]. Tento dokument neobsahuje žádný příklad a i když obsahuje po formální stránce všechny potřebné informace, velmi špatně se v něm orientuje a troufám si tvrdit, že čistě na základě něj dokáže cokoliv nakonfigurovat jen málokdo.

1.3 Struktura této práce.

Tato práce by tedy měla být manuálem, který vysvětluje principy správy fyzických a IP adres, směrovacích tabulek a v menší míře také samotných síťových rozhraní. První část vysvětluje jakým způsobem toto provést přímo přes Netlink socket. Ve druhé části je potom zdokumentován balíček, který umožňuje totéž, ale jednodušším vysokoúrovňovým způsobem.

Chtěl bych na úvod poznamenat, že téměř vše se nastavuje pomocí konstant. V práci uvádím téměř vždy pouze nejběžněji používané hodnoty. Kompletní výčet je obsažen v modulu constants, balíčku PyNetlink.

2 Správa IP protokolu přes Netlink

Tato kapitola popisuje jakým způsobem navázat komunikaci přes Netlink a strukturu odesílaných dat. Dále také jak konfigurovat některá nastavení IP subsystému.

2.1 Vytvoření komunikačního rozhraní (socketu)

Komunikace přes Netlink socket je velmi podobná práci se soubory nebo jakýmkoli jiným socketem.

Je potřeba vytvořit socket rodiny AF_NETLINK, a jako typ socketu nastavit buď SOCK_RAW a nebo SOCK_DGRAM. Netlink mezi nimi nerozlišuje.

```
import socket
sock = socket.socket(socket.AF_NETLINK, socket.SOCK_RAW)
```

Výpis 1: Vytvoření socketu.

Volání socket() v Pythonu vrátí objekt reprezentující socket. Pro odesílání a příjem dat slouží metody send() a recv().

2.2 Obecný formát všech zpráv - Hlavička NlMsgHdr

Každá zpráva zasílaná nebo přijatá přes Netlink socket, se skládá většinou z více částí. Každá část má svoji hlavičku. Za ní může následovat buď přímo zpracovatelný datový typ, například číslo ve formátu int, nebo textový řetězec a nebo hlavička další zanořené části. Jednotlivé části tak v podstatě tvoří stromovou strukturu.

Tady je důležité, že každá jednotlivá část musí mít délku v bytech dělitelnou číslem čtyři. Pokud tomu tak není, doplní se na konec této části výplň v podobě nulových bytů tak aby tato podmínka byla splněna. V hlavičce této části bude pole označující její délku obsahovat délku bez výplně. Naopak při výpočtu délky nadřazené části se vnořené části berou jako celek i včetně výplně. Tak jak to ukazuje tabulka 1.

Aby bylo možné identifikovat a vůbec dále zpracovávat odeslaná nebo přijatá data jak na straně jádra tak aplikací v uživatelském prostoru, musí každá zpráva začínat hlavičkou NlMsgHdr. Strukturu této hlavičky zázorňuje tabulka 2. Význam jednotlivých polí v této je vysvětlen v následující výčtu:

- **nlmsg_len:** Délka celé zprávy v bytech v četně hlavičky.
- **nlmsg_type:** Toto pole slouží k identifikaci dat za hlavičkou.
- **nlmsg_flags:** Příznaky, jinak také vlaječky.
  - **NLM_F_REQUEST = 1:** Tato zpráva je požadavek. Tento příznak by měl být nastaven u všech zpráv posílaných do jádra.
  - **NLM_F_ACK = 4:** Požadavek na potvrzení. Více v kapitole 2.5
  - **NLM_F_ECHO = 8:** Podle RFC [3] by zpráva odeslaná s tímto příznakem měla být obratem přeposlána zpět. V praxi se mi toto zprovoznit nepodařilo

4B	HlavičkaA délka=20		
4B		HlavičkaB délka=7	
4B			data 3B výplň 1B
4B		HlavičkaC délka=5	
4B			data 1B výplň 3B

Tabulka 1: Příklad zanořování a zarovnávání jednotlivých částí zprávy.

Název	Délka	Znaménko
nlmsg_len	32b	ne
nlmsg_type	16b	ne
nlmsg_flags	16b	ne
nlmsg_seq	32b	ne
nlmsg_pid	32b	ne

Tabulka 2: Struktura NlMsgHdr.

  - **NLM_F_MULTI = 2:** Za zprávou s tímto příznakem bude následovat další zpráva s odpovědí.
  - **NLM_F_ROOT = 0x100:** Máme zájem o kompletní výpis.
  - **NLM_F_MATCH = 0x200:** Odpověď bude obsahovat jenom položky odpovídající filtračnímu kritériu.
  - **NLM_F_ATOMIC = 0x400:** Data pro odpověď by měla být shromážděna atomicky - aby byla odpověď konzistentní.
  - **NLM_F_DUMP:** Kombinace příznaků NLM_F_ROOT a NLM_F_MATCH.
  - **NLM_F_REPLACE = 0x100:** Nahradit prvek.
  - **NLM_F_EXCL = 0x200:** Nenahrazovat prvek pokud existuje.
  - **NLM_F_CREATE = 0x400:** Vytvořit nový prvek.
  - **NLM_F_APPEND = 0x800:** Připojit prvek na konec.

- **nlmsg_pid:** Identifikační číslo procesu popřípadě procesu a vlákna. Slouží pouze k identifikaci odpovědí. V tomto poli může být v podstatě libovolná hodnota. V odpovědi bude toto pole nastavené totožně. Toto se hodí v případě že k jednomu socketu přistupuje najednou více jak jedno vlákno či proces.

- **nlmsg_seq:** Toto pole má podobný význam jako pole předchozí. Umožňuje přiřadit ke kterému dotazu patří daná odpověď.

Některé typy zpráv:

- **RTM_NEWLINK, RTM_DELLINK, RTM_GETLINK, RTM_SETLINK:** Správa síťových rozhraní

Formát	Odpovídající typ v C	Typ v Pythonu	Velikost v bytech
x	pad byte	bez hodnoty	1
c	char	bytes délky 1	1
b	signed char	int	1
B	unsigned char	int	1
h	short	int	2
H	unsigned short	int	2
i	int	int	4
I	unsigned int	int	4

Tabulka 3: Některé možné znaky formátovacího řetězce třídy Struct.

- **RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR:** Správa IP adres.
- **RTM_NEWROUTE, RTM_DELROUTE, RTM_GETROUTE:** Správa směrovacích tabulek.
- **NLMSG_NOOP = 1:** Zpráva tohoto typu bude ignorována.
- **NLMSG_ERROR = 2:** Zpráva je oznámení o chybě. Více v části 2.5.
- **NLMSG_DONE = 3:** Ukončení sekvence složené z více zpráv.

2.3 Zpracování binárních dat v Pythonu

Komunikace přes socket je binární. V Pythonu jsou tato data reprezentována datovým typem bytes, který je neměnitelný, popřípadě měnitelným typem bytearray. A protože, na rozdíl například od C++, Python nemá pevně stanovenu délku proměnných, obsahuje standardní knihovna Pythonu modul struct. Tento modul obsahuje stejnojmennou třídu Struct(), které se jako jediný argument předá textový řetězec reprezentující formát dat. Nejpoužívanější znaky, ze kterých je možné tento řetězec složit, ukazuje tabulka 3. Kompletní výčet je v dokumentaci [4].

Převod proměnných na bytes se provede zavoláním metody pack(). Do argumentu je třeba vypsat všechny proměnné v odpovídajícím pořadí. Počet proměnných musí přesně odpovídat požadovanému formátu.

K obrácenému převodu slouží metoda unpack(), které se do argumentu předá řetězec bytes. I tady musí délka tohoto řetězce odpovídat délce formátu v bytech. Délku je možné zjistit pomocí property size.

Následující kód převádí dvě proměnné, které odpovídají typu unsiged short na bytes a zpět.

```
from struct import Struct

promA = 1
promB = 2
```

```
s = Struct('HH')
bst = s.pack(promA, promB)
print("bst=", bst)

(pA, pB) = s.unpack(bst[:s.size])
print(pA, pB)
```

Výpis 2: Práce s binárními daty.

S bytes se pracuje jako s jakýmkoliv jiným řetězcem nebo seznamem. Například dost často je třeba ořezávat data na potřebnou délku. V předchozím kódu jsem to vyřešil takto: bst[:s.size]

Při větším počtu zpracovávaných proměnných je dost nepraktické je neustále vypisovat. Můžeme si pomoct kolekcí namedtuple z balíčku collections, který je standardní součásti Pythonu, a operátorem rozbalení $*$. Tato kolekce funguje stejně jako tuple, ale s tím rozdílem, že jednotlivé proměnné jsou pojmenované a je třeba nejprve vytvořit konstruktor. Tento nám vrátí funkce namedtuple(jmeno, format). Parametr jmeno je libovolný řetězec, kterým nově vytvořené objekty pojmenujeme. Parametr format je jeden řetězec s názvy proměnných, oddělenými bílým znakem.

Předchozí příklad by s využitím namedtuple vypadal takto:

```
from struct import Struct
from collections import namedtuple

Ntuple = namedtuple('Ntuple', 'prom1 prom2')
proms = Ntuple(1, 2)

s = Struct('HH')
bst = s.pack(*proms)
print("bst=", bst)

p = Ntuple(*s.unpack(bst[:s.size]))
print(*p)
```

Výpis 3: Práce s binárními daty s využitím namedtuple.

K převodu řetězců z bytes na typ str a zpět slouží metoda decode() typu bytes a nebo encode() volaná nad objektem typu str.

2.4 Odesílání a příjem dat přes Netlink socket

Odeslání dat do jádra je velmi jednoduché. Provede se pomocí zavolání metody send() vytvořeného socketu. Jako argument se jí předá řetězec datového typu bytes.

Příjem dat se realizuje zavoláním metody recv() vytvořeného socketu. Tato metoda má jediný argument a to maximální délku dat, která přečte. Tato metoda čeká dokud jí jádro nepošle zprávu, kterou by mohla přijmout a zablokuje tak provádění vlákna. K ovlivnění tohoto chování Python nabízí hned několik metod. Asi nejuniverzálnější je metoda settimeout(). Má pouze jediný argument, který vyjadřuje počet vteřin typu float, jak dlouho bude metoda recv() čekat na příchozí zprávu. Pokud se do tohoto limitu nic

Název	Délka	Znaménko
Číslo chyby	32b	ano

Tabulka 4: Struktura chybové zprávy.

nepříjme, dojde k vyvolání výjimky timeout. Nulová hodnota tohoto argumentu znamená, že pokud v socketu nečekají žádné zprávy v době volání metody recv(), dojde k vyvolání výjimky error. Chceme-li přejít zpět do blokujícího režimu, zavoláme metodu settimeout() s argumentem None.

Tady bych chtěl upozornit na to, že jde o paketový protokol. To v praxi znamená, že jedno zavolání recv() přijme jednu celou zprávu. Pokud je tato zpráva delší než stanovené maximum, ořízne se a zbytek je nenávratně ztracen. Další zavolání této metody vrátí až následující zprávu.

Dotazy a zejména odpovědi se mohou skládat z více než jedné zprávy. V tomto případě mají zprávy nastaven příznak NLM_F_MULTI = 2. To znamená, že za právě přijatou zprávou bude následovat další. Poslední zpráva tento příznak nastaven nemá a je typu NLM_F_DONE = 2. Nenese už žádná jiná data.

2.5 Rozpoznání chybových stavů

Zejména první pokusy s Netlinkem zpravidla vedou k chybovým odpovědím. Netlink chyby oznamuje, stejně jako jakoukoliv jinou odpověď, zasláním zprávy s velmi jednoduchou strukturou. Standardní hlavička s nastaveným polem typ na NLMSG_ERROR = 2 je následovaná jedním 4B dlouhým polem typu integer, které obsahuje číselný kód chyby. Dále následuje hlavička NlMsgHdr původní zprávy.

Pro překlad těchto stavů, do člověku srozumitelné podoby, nabízí Python funkci strerror () z modulu os. Vrací řetězec popisující chybu na základě jejího čísla předaného v argumentu. Očekává ale kladné chybové kódy, proto je v příkladu4 použita funkce abs().

```
import os

errno = -1
print(os.strerror (abs(errno)))
```

Výpis 4: Čtení chybových zpráv.

Jedinou výjimkou je kód chyby 0. Toto neznamená, že došlo k chybě, ale jedná se o takzvaný ack - potvrzovací zprávu. O potvrzení jakékoliv zaslané zprávy je možné požádat nastavením příznaku NLM_F_ACK = 4.

2.6 Praktická ukázka základní komunikace

Informace, obsažené v předchozích částech této práce, by měly být dostatečné na ověření funkčnosti komunikace s jádrem přes Netlink socket. Pokusím se je tedy shrnout na praktické ukázce. Je velmi jednoduchá. Zašleme zprávu s požadavkem na její potvrzení, které následně přečteme.

V první části je potřeba naimportovat potřebné třídy a konstanty. Dále si připravíme standardní hlavičku. A to jak namedtuple, který nám bude udržovat samotné údaje v hlavičce a usnadňovat jejich výpis a čtení tak i objekt struct, který nám umožní ji převádět z a do formátu bytes.

```
from socket import socket, SOCK_DGRAM, AF_NETLINK
from struct import Struct
from collections import namedtuple

sNlMsgHdr = Struct("IHHII")
TNlMsgHdr = namedtuple("NlMsgHdr", "len type flags seq pid")
```

Výpis 5: Příprava hlavičky NlMsgHdr pomocí standardních nástrojů.

Dále si vytvoříme namedtuple, tak aby odpovídal struktuře hlavičky. Po té vytvoříme zprávu s typem NLMSG_NOOP=1. Tato zpráva bude jádrem ignorována, což se nám hodí. V tomto příkladně nechceme nic konfigurovat. Pole flags této zprávy bude obsahovat pouze příznak NLM_F_ACK=4. Na základě tohoto příznaku nám jádro odešle potvrzovací zprávu. Poté už zbývá jenom vyplnit délku zprávy do pole len. Za hlavičku nebudeme nic dalšího připojovat, takže délka naší zprávy je rovna délce naší hlavičky. Tu zjistíme následovně sNlMsgHdr.size. Na hodnotách polí seq a pid v tomto případě nezáleží.

namedtuple s hlavičkou následně převedeme pomocí třídy Struct na bytes. Poté vytvoříme socket a hlavičku přes něj odešleme.

```
hlavicka = TNlMsgHdr(len=sNlMsgHdr.size, type=1, flags=4, seq=1, pid=0)

s = socket(AF_NETLINK, SOCK_DGRAM)
s.send(sNlMsgHdr.pack(*hlavicka))
```

Výpis 6: Odeslání hlavičky NlMsgHdr pomocí standardních nástrojů.

Nyní přečteme odpověď a socket uzavřeme, nebude už potřeba. Z odpovědi odsekneme část odpovídající hlavičce. Tuto část poté převedeme do namedtuple a rovnou vytiskneme.

```
odpoved = s.recv(2048)
s.close()

hlavicka = TNlMsgHdr( *sNlMsgHdr.unpack(odpoved[:sNlMsgHdr.size]))
odpoved = odpoved[sNlMsgHdr.size:]
print(hlavicka)
```

Výpis 7: Přijetí odpovědi pomocí standardních nástrojů.

V tuto chvíli by měl skript vypsat na standardní výstup následující:
NlMsgHdr(len=36, type=2, flags=0, seq=1, pid=7905)
Vidíme hlavičku odpovědi. Nejvíce nás bude zajímat typ zprávy. Jak je popsáno v části 2.2, typ 2 znamená chybovou zprávu. Nyní nás bude zajímat o jakou chybu jde a jestli je to vůbec chyba. Vypíšeme tedy další 4byty zprávy.

```
errnoBytes = odpoved[:4]
odpoved = odpoved[4:]
```

Název	Délka	Znaménko
ifa_len	16b	ne
ifa_type	16b	ne

Tabulka 5: Struktura NlAttr.

```
print([errnoBytes])
```

Výpis 8: Výpis chybového kódu.

Výstup této části skriptu by měl vypadat takto: [b'x00x00x00x00']. Už z pohledu na řetězec bytes je patrné, že hodnota chybového kódu je nula. Tedy nejde o chybu, ale o potvrzení, které jsme si vyžádali.

Ve funkci **print** jsem uzavřel proměnou s řetězcem bytes do hranatých závorek jako seznam. Jinak by se řetězec považoval přímo za text a výpis by byl nečitelný.

2.7 Obecný atribut - Struktura NlAttr

Tato struktura je využívaná u většiny odpovědí i požadavků. Z pravidla se využívají pro přenášení dat, která nemusí být přenášena vždy a nebo přenáší data proměnné délky. Například textové řetězce nebo IP adresy. V RFC3549 [3] jsou tyto atributy označeny jako TVL.

Jak je patrné z tabulky 5, tato hlavička má velmi jednoduchou strukturu.

- **nla_len:** Délka přenášených dat i s hlavičkou. (Nezahrnuje případnou výplň.)
- **nla_type:** Identifikace nesených dat.

Textové řetězce jsou, stejně jako například v C, ukončeny nulovým bytem. Tento byte je také součástí řetězce a počítá se do délky v poli ifa_len . Text ”eth0”By tedy měl toto pole nastaveno na 4B hlavičky + 5B text včetně ukončovacího bytu = 9B. Data připojená za hlavičkou by vypadala následovně:

'e'	't'	'h'	'0'	'\0'	výplň		
0x65	0x74	0x68	0x30	0x00	0x00	0x00	0x00

2.8 Žádosti o výpisy - Struktura Family

Než se dostanu k podrobnějšímu popisu správy síťového subsystému, chtěl bych se zde zmínit obecně o žádosti, o výpis čehokoliv (rozhraní, IP adresy,...). Zpráva s požadavkem se totiž liší pouze typem zprávy.

V hlavičce zprávy se vyplní typ RTM_GET$*$, který odpovídá tomu co chceme vypsat. Dále pak příznaky se nastaví na následující logický součet konstant

NLMF_F_REQUEST|NLM_F_DUMP. Za hlavičku se poté připojí čtyři bajty dat. Poslední tři byty nemají význam a měly by být nulové. První byt je možné ponechat buď nulový,

Název	Délka	Znaménko
ifi_family	8b	ne
ifi_pad	8b	ne
ifi_typ	16b	ne
ifi_index	32b	ano
ifi_flags	32b	ne
ifi_change	32b	ne

Tabulka 6: Struktura IfInfoMsg.

to znamená, že máme zájem o vše, a nebo může nabývat hodnot konstant AF_INET nebo AF_INET6. Potom obdržíme pouze odpovědi odpovídající rodině IPv4 nebo IPv6.

Poslední dvě zmíněné konstanty obsahuje modul socket.

2.9 Práce se síťovými rozhraními.

2.9.1 Hlavička IfInfoMsg

Tato hlavička je součástí každé zprávy, která se týká správy síťových rozhraní. Její strukturu znázorňuje tabulka 6.

- **ifi_family:** Rodina zařízení. Nikdy jsem se nesetkal s jinou než AF_UNSPEC=0.
- **ifi_pad:** Výplň. Nenese žádnou informaci. Měla by mít hodnotu 0.
- **ifi_type:** Typ zařízení. Nejběžněji:
  - **ARPHRD_LOOPBACK = 772:** Zařízení typu loopback.
  - **ARPHRD_ETHER = 1:** Rozhraní typu Ethernet. Nejběžnější hodnota.
- **ifi_index:** Číselná identifikace zařízení v jádře.
- **ifi_change:** I když se v některých pramenech píše, že toto pole je rezervováno pro budoucí použití, využívá se při požadavku na změnu příznaků a to tak, že se v tomto poli nastaví na jedničku ty bity, u kterých se má provést změna.
- **ifi_flags:** Příznaky signalizující stav zařízení. Nejvíce nás budou zajímat asi následující:
  - **IFF_UP = 0x1:** Administrativní status rozhraní.
  - **IFF_BROADCAST = 0x2:** Na rozhraní je platná broadcast adresa.
  - **IFF_LOOPBACK = 0x8:** Jde o síť typu loopback.
  - **IFF_RUNNING = 0x40:** Rozhraní je aktivní a zároveň je v pořádku nosná.
  - **IFF_PROMISC = 0x100:** Zařízení je v promiskuitním módu - přijímá všechny pakety.

NlMsgHdr		
	IfInfoMsg	
		RtAttr
		RtAttr
		...

Tabulka 7: Rozvrstvení informací o síťovém rozhraní.

- **IFF_LOWER_UP = 0x10000:** Ovladač hlásí, že je aktivní i na první vrstvě - připojen kabel.
- **IFF_DORMANT = 0x20000:** Zařízení čeká na nějakou externí akci aby se mohlo stát aktivním. Například modem při vytáčení čísla.

2.9.2 Získání informací o všech rozhraních

K získání informací o všech rozhraních je potřeba zaslat žádost, tak jak jsem ji popsal v kapitole 2.8. Jako typ se použije RTM_GETLINK = 18. Odpověď se bude skládat zpravidla z několika zpráv se strukturou jako v tabulce 7.

Dost zajímavých informací nese samotná struktura IfInfoMsg. V první řadě je to hodnota v poli ifi_index . Je to číselná jednoznačná identifikace zařízení v jádře. Při konfiguraci záznamů směrovací tabulky a nebo IP adres se na zařízení odkazuje právě přes tuto hodnotu. Informace o stavu zařízení najdeme v poli ifi_flags struktury IfInfoMsg. Jejich významy jsou popsány v kapitole 2.9.1 na straně 12.

Za touto strukturou následuje řada struktur RtAttr. Jsou totožné se strukturou NlAttr, popsanou v kapitole 2.7. Nasledující výčet popisuje některé typy těchto atributů:

- **IFLA_ADDRESS = 1:** Hardwarová adresa zařízení.
- **IFLA_IFNAME = 3:** Název zařízení ve formě textového řetězce. Více o textových řetězcích v kapitole 5
- **IFLA_MTU = 4:** Číslo bez znaménka dlouhé 4B, které vyjadřuje maximální velikost zprávy, kterou jde přes rozhraní odeslat - MTU.
- **IFLA_QDISC = 6:** Textový řetězec se jménem queuing discipline - určuje jakým způsobem se prioritizují a řadí do front odesílané rámce. přes toto rozhraní.
- **IFLA_BROADCAST = 2:** Adresa pro všesměrové vysílání na druhé vrstvě.

2.9.3 Konfigurace rozhraní

Konfigurační zpráva má stejnou strukturu jako zprávy s informacemi o rozhraní. Viz. tabulka 7. V hlavičce NlMsgHdr se nastaví typ zprávy na RTM_NEWLINK=16 a nastaví se příznakový bit NLM_F_REQUEST=1. Za touto hlavičkou následuje hlavička IfInfoMsg. Zde je nutné vyplnit index zařízení, aby bylo jasné, které rozhraní se konfiguruje.

Název	Délka	Znaménko
ifa_family	8b	ne
ifa_prefixlen	8b	ne
ifa_flags	8b	ne
ifa_scope	8b	ne
ifa_index	32b	ne

Tabulka 8: Struktura IfAddrMsg.

Pokud chceme změnit některý příznakový bit v této hlavičce, například chceme-li zařízení aktivovat, nastavíme bit na pozici odpovídající hodnotě konstanty IFF_UP=1 na odpovídající hodnotu tj. 1 v poli ifi_flags . Zároveň je nutné také nastavit na jedničku bit na stejné pozici v poli ifi_change. Ostatní příznakové bity takto nebudou změněny.

Za tuto hlavičku se následně mohu připojit struktury RtAttr pokud požadujeme změnu některých dalších atributů.

2.10 Správa IP adres na rozhraních

2.10.1 Hlavička IfAddrMsg

Toto je obecná hlavička každé jedné IP adresy. Její strukturu popisuje tabulka 8.

Popis polí ve struktuře:

- **ifa_family:** Rodina adresy.
  - **AF_INET = 2:** IPv4 adresa.
  - **AF_INET6 = 10:** IPv6 adresa.
- **ifa_prefixlen:** Maska IP adresy vyjádřená jako počet jedniček v masce.
- **ifa_flags:** Příznaky IP adresy:
  - **IFA_F_SECONDARY = 0x01:** Jde o sekundární adresu. Každé rozhraní na jedné podsíti může mít jednu hlavní a libovolné množství sekundárních adres. Smazáním primární adresy dojde k vymazání všech adresa na této podsíti. Smazáním sekundární adresy nebudou zbylé adresy dotčeny.
  - **IFA_F_PERMANENT = 0x80:** Trvalá adresa vytvořena uživatelem
- **ifa_scope:** Oblast platnosti adresy:
  - **RT_SCOPE_UNIVERSE = 0:** Adresa platná všude
  - **RT_SCOPE_SITE = 200:** Adresa platná pouze v rámci sítě. (Pouze u IPv6)
  - **RT_SCOPE_LINK = 253:** Adresa platná pouze v rámci rozhraní.
  - **RT_SCOPE_HOST = 254:** Adresa platná pouze v rámci stroje.
  - **RT_SCOPE_NOWHERE = 255:** Adresa není platná nikde.
- **ifa_index:** Index rozhraní ke kterému adresa patří.

NlMsgHdr		
	NlAddrMsg	
		RtAttr
		RtAttr
		...

Tabulka 9: Rozvrstvení výpisu IP adres.

2.10.2 Získání všech adres

Tak jako je popsáno v kapitole 2.8, získáme všechny adresy jako odpovědi na žádost kde se jako typ v NlMsgHdr nastaví hodnota RTM_GETADDR=22. Je možné požádat si pouze o určitou rodinu adres a to nastavením struktury family na hodnotu AF_INET=2 pro adresy IPv4 a nebo AF_INET6=10 pro adresy rodinu IPv6.

Struktura každé odpovědi je prakticky stejná jako u výpisu rozhraní. Popisuje jej tabulka 9.

Hlavička IfAddrMsg nese v první řadě informaci o rodině adresy v poli ifa_family . Dále pak masku podsítě v poli ifa_prefixlen a rozsah platnosti adresy v poli ifa_scope.

Za hlavičkou následují struktury NlAttr s dalšími daty. Zejména samotnou adresou. Mimo jiné mohou být zejména následujících typů:

- **IFA_ADDRESS = 1:** Samotná IP adresa.
- **IFA_LOCAL = 2:** Používá u spojů typu bod-bod, kdy obsahuje adresu na zařízení. Pole IFA_ADDRESS potom obsahuje adresu protistrany.
- **IFA_BROADCAST = 4:** Adresa pro broadcast - všesměrové vysílán.í

2.10.3 Přidání adresy na rozhraní

Podobně jako při konfiguraci síťových rozhraní je potřeba sestavit hlavičku NlMsgHdr s typem RTM_NEWADDR=20 a nastavit příznakový bit NLM_F_REQUEST=1 a NLM_F_CREATE . Pokud chceme tuto adresu přidat k dalším adresám na podsíti, je třeba nastavit i bit NLM_F_EXCL. Jinak by přidávaná adresa nahradila existující adresy na této podsíti.

Za touto hlavičkou následuje hlavička IfAddrMsg. Zde se vyplní rodina adresy do pole ifa_family , do pole ifa_prefixlen maska podsítě a do pole ifa_scope rozsah platnosti adresy. Nejspíše RT_SCOPE_UNIVERSE=0.

Nakonec je nutné připojit minimálně atributy NlAttr typu IFA_LOCAL a IFA_ADDRESS. Za každý z nich se připojí samotná IP adresa.

2.10.4 Smazání adresy

Adresa se maže stejným způsobem jako se přidává. Jediným rozdílem je typ v hlavičce NlMsgHdr, který se nastaví na hodnotu RTM_DELADDR. I tady je nutné připojit na konec dva atributy, jako v předchozím případě.

Název	Délka	Znaménko
rtm_family	8b	ne
rtm_dst_len	8b	ne
rtm_src_len	8b	ne
rtm_tos	8b	ne
rtm_table	8b	ne
rtm_protocol	8b	ne
rtm_scope	8b	ne
rtm_type	8b	ne
rtm_flags	32b	ne

Tabulka 10: Struktura hlavičky RtMsg.

2.11 Správa pravidel ve směrovacích tabulkách

Princip správy pravidel ve směrovací tabulce je velmi podobný správě IP adres. Drobnou komplikací je, že je potřeba rozlišovat jestli má směrovací pravidlo jeden cíl a nebo má cílů více (tzv. multipath routing). Každý z těchto druhů má jinou skladbu atributů. Je to určitá daň za univerzálnost síťového subsystému jádra.

Dále bych chtěl upozornit na skutečnost, že směrovací pravidlo nemusí mít vždy cílovou IP adresu. Má pouze cílové rozhraní. Takové pravidlo říká, že toto rozhraní patří přímo do této podsítě. Jádro si takováto pravidla vytváří samo pokaždé, když je na některé rozhraní přidána IP adresa.

2.11.1 Hlavička RtMsg

Tak jako v předchozích dvou případech, i při správě směrování je standardní hlavička zprávy NlMsgHdr následována právě touto hlavičkou se strukturou znázorněnou tabulkou 10. Za ní následují atributy s hlavičkami RtAttr. Hlavička RtAttr má strukturu shodnou jako hlavička NlAttr popsaná v části 2.7 na straně 11. Význam jednotlivých polí této hlavičky je následující:

- **rtm_family:** Rodina směrovacího pravidla. Stejně jako u IP adresy může nabývat hodnot AF_INET=2 pro IPv4 a nebo AF_INET6=10 pro IPv6.
- **rtm_dst_len:** Maska podsítě do které má provoz dorazit
- **rtm_src_len:** Nesetkal jsem se s jinou hodnotou než 0
- **rtm_tos:** Identifikace druhu služby provozu, pro který bude toto pravidlo použito. Pokud je nula, použije se pro veškerý provoz.
- **rtm_table:** Číslo tabulky do které směrovací pravidlo patří
  - **RT_TABLE_UNSPEC = 0:** Tabulka není specifikována.
  - **RT_TABLE_MAIN = 254:** Hlavní směrovací tabulka.

Název	Délka	Znaménko
rtm_len	16b	ne
rtm_flags	8b	ne
rtm_hops	8b	ne
rtm_ifindex	32b	ne

Tabulka 11: Struktura hlavičky RtNextHop.

  - **RT_TABLE_LOCAL = 255:** Tabulka s pravidly pro podsítě, ve kterých se počítač přímo nachází.

- **rtm_protocol:** Označuje jakým způsobem pravidlo vzniklo. Následující výčet není konečný. Mohou být použity i jiné konstanty například pro pravidla, která do tabulky vložil směrovací démon.
  - **RTPROT_UNSPEC = 0:** Způsob vzniku nebyl specifikován.
  - **RTPROT_KERNEL = 2:** Pravidlo bylo vytvořeno jádrem.
  - **RTPROT_BOOT = 3:** Pravidlo vzniklo během startu systému.
  - **RTPROT_STATIC = 4:** Statické pravidlo vytvořené administrátorem
- **rtm_scope:** Oblast platnosti pravidla
  - **RT_SCOPE_UNIVERSE = 0:** Platné všude
  - **RT_SCOPE_SITE = 200:** Platné pouze v rámci sítě. (Pouze u IPv6)
  - **RT_SCOPE_LINK = 253:** Platné pouze v rámci rozhraní.
  - **RT_SCOPE_HOST = 254:** Platné pouze v rámci stroje.
  - **RT_SCOPE_NOWHERE = 255:** Není platné nikde.
- **rtm_type:** Typ pravidla
- **rtm_flags:** Příznaky.

2.11.2 Hlavička RtNextHop

Jak už jsem předesílal výše, směrovací záznam nemusí mít pouze jeden cíl. Pokud jich má více, používá se k popsání každého z cílů tato hlavička. Její strukturu znázorňuje tabulka 11.

- **rtm_len:** Délka dat včetně této hlavičky
- **rtm_flags:** Příznaky. Většinou je toto pole nulové
- **rtm_hops:** Toto polo označuje váhu cesty. Čím větší váha, tím častěji bude tato cesta využívána.
- **rtm_ifindex:** Číselný index zařízení.

NlMsgHdr			
	RtMsg		
		NlAttr	
		NlAttr	
			RtNextHop
				NlAttr
			RtNextHop
				NlAttr
		NlAttr	

Tabulka 12: Rozvrstvení informací popisující směrovací pravidlo.

2.11.3 Získání všech pravidel ze všech směrovacích tabulek

Získání všech záznamů, ze všech směrovacích tabulek, se provede jako vždy odesláním žádosti popsané v kapitole 2.8 na straně 11 s polem nlmsg_type nastaveným na hodnotu RTM_GETROUTE=26. Bohužel jedinou možností filtrace zpráv na straně jádra je nastavení rodiny směrovacích pravidel. Odpověď poté bude obsahovat pouze pravidla typu IPv4 nebo IPv6. Nepodařilo se mi přijít na způsob, jak si požádat o pravidla pouze z jedné směrovací tabulky a s největší pravděpodobností to ani možné není.

Příklad struktury jedné odpovědi je znázorněn v tabulce 12. Za hlavičkou NlMsgHdr vždy následuje hlavička RtMsg. Dále následují atributy. Jejich strukturu popisuje kapitola 2.7 na straně 11. Některé typy těchto atributů popisuje následující výčet:

- **RTA_DST = 1:** IP adresa podsítě kam se má provoz doručit.
- **RTA_OIF = 4:** 4B dlouhé číslo bez znaménka, určující rozhraní, přes které se bude provoz posílat dále.
- **RTA_PRIORITY = 6:** 4B dlouhé číslo bez znaménka určující metriku. Tj. vzdálenost k cíli. Pokud se v tabulce nachází více pravidel, použije se to s nižší metrikou.
- **RTA_GATEWAY = 5:** IP adresa dalšího skoku kam se má provoz předávat.
- **RTA_MULTIPATH = 9:** Za hlavičkou tohoto atributu následuje několik dalších hlaviček RtNextHop. Bude vysvětleno dále.

Tabulka 12 ukazuje nejsložitější případ struktury odpovědi. Pravidlo zde má atribut, ve kterém je zanořeno několik cílů. Je vždy typu RTA_MULTIPATH=9 a obsahuje několik hlaviček RtNextHop se strukturou popsanou v části 2.11.2. Každá tato hlavička nese informace o váze cesty a indexu odchozího rozhraní. Za hlavičkou může následovat atribut typu RTA_GATEWAY=5 s IP adresou zařízení, na které se má provoz přeposílat.

V případě že má směrovací pravidlo pouze jeden cíl, je za hlavičkou RtMsg atribut typu RTA_OIF. Za ním následuje index odchozího rozhraní jako 4B dlouhé číslo bez znaménka. Případnou cílovou IP adresu zařízení, na které se má provoz přeposílat, obsahuje atribut typu RTA_GATEWAY=5.

Pokud nejde o směrovací pravidlo určující výchozí bránu, najdeme v odpovědi také atribut typu RTA_SRC=2. Za ním následuje IP adresa podsítě, do které se má provoz doručit. Maska podsítě je přímo v hlavičce RtMsg v poli rtm_dst_len.

2.11.4 Vložení záznamu do směrovací tabulky

Jako vždy je nutné sestavit hlavičku NlMsgHdr s typem RTM_NEWROUTE=24 a příznaky NLM_F_CREATE, NLM_F_REQUEST a NLM_F_EXCL.

Za touto hlavičku se připojí hlavička RtMsg. Zde je potřeba nastavit pole rtm_table na číslo požadované směrovací tabulky. Například pro hlavní tabulku RT_TABLE_MAIN=254. Pokud nevkládáme výchozí bránu, je nutné také vyplnit do pole rtm_dst_len masku podsítě. Pole rtm_ifindex je třeba vyplnit jenom pokud pravidlo nebude mít nastavenu IP adresu, na kterou se bude provoz přeposílat. V opačném případě je možné do tohoto nastavit nulu. Jádro si zařízení doplní samo. Ostatní pole je možné taktéž ponechat nastavená na nulu, pokud nemáme zájem je nějak konkrétně specifikovat.

Za hlavičku RtMsg se mohou připojit další atributy. Zejména ty co jsou popsány v předchozí části.

2.11.5 Odstranění pravidla

Pravidlo ze směrovací tabulky se odstraní téměř stejným způsobem, jako když se přidávalo. Jediným rozdílem je hlavička NlMsgHdr, kde se jako typ použije RTM_DELROUTE=25 a nastaví se pouze příznak NLM_F_REQUEST.

3 Balíček PyNetlink

Tak jak požaduje zadání, vytvořil jsem balíček, který zjednodušuje několik základních administračních činností prováděných přes Netlink socket. Balíček jsem se snažil napsat tak, aby byl snadno čitelný a v budoucnu umožňoval co nejednodušší rozšiřování, bez zbytečného opakování kódu. Vzhledem k tomu že má jít o vysokoúrovňovou knihovnu, Není třeba při práci nikterak uvolňovat systémové zdroje voláním metody close() nebo čímkoliv podobným a taktéž jsem se snažil nezatěžovat uživatele/vývojáře řešením věcí, které za něj může vyřešit počítač, takže například není nutné přímo specifikovat rodinu požadované IP adresy.

Než se dostanu dále, tak bych rád vysvětlil pojem properties protože ho budu docela často používat. Property je sada dvou metod, které se ale navenek tváří jako běžné atributy třídy. Na rozdíl od atributů ale mohou být omezeny jakou pouze pro čtení nebo pouze pro zápis. Český překlad by mohl znít vlastnost, ale anglický originál mi přijde více zažitý tak zůstanu u něj.

3.1 Třída MacAddress

Tato třída je velmi jednoduchá. Slouží pouze k přenosu fyzické adresy síťového rozhraní. Konstruktor očekává jediný parametr a to řetězec bytes v délce šest. Druhou možností je využít statické metody fromString(), která vrací objekt MacAddress vytvořený na základě textové reprezentace této adresy. U obou způsobů dojde k vyvolání výjimky ValueError v případě špatných dat v argumentech.

Objekt je také možné převést pomocí funkcí bytes() nebo str () na stejnojmenné datové typy.

3.2 Třídy IpAddress, IpNet a IpNetDetailed

Všechny tyto třídy slouží k zapouzdření IP adresy. Stejně jako v třídě MacAddr, konstruktor zde počítá s řetězcem typu bytes. Rodina adresy se rozpozná automaticky na základě délky tohoto řetězce. Objekty všech tří tříd je možné vytvářet z běžné textové reprezentace IP adres, pomocí metody fromString. I tato metoda rozpozná rodinu adresy z formátu řetězce.

IpAddress:

Toto je základní třída. Reprezentuje pouze samotnou IP adresu bez masky. Má následující metody a properties:

- **__init__(_bytes):** Konstruktor, který v argumentu očekává řetězec bytes. Jeho délka musí být 4 pro IPv4 adresu nebo 16 pro IPv6 adresu. Jinak dojde k vyvolání výjimky ValueError.
- **fromString(string):** Statická metoda. Vrací instance tohoto objektu na základě její textové reprezentace předané v argumentu. Například "192.0.2.1"nebo "2001:DB8::1". Pokud se převod nezdaří dojde k vyvolání výjimky ValueError.

- **__bytes__():** Umožňuje převod na bytes pomocí stejnojmenné funkce bytes().
- **__int__():** Umožňuje převod na int pomocí stejnojmenné funkce int().
- **__len__():** Umožňuje zjištění počtu bytů adresy pomocí funkce len().
- **__str__():** Umožňuje převod na textový řetězec pomocí funkce str().
- **__eq__():** Přetížení operátoru rovnosti ==.
- **family:** Property vracející konstanty AF_INET nebo AF_INET6 na základě rodiny adresy.

IpNet:

Tato třída reprezentuje IP adresu i v četně masky podsítě. Dědí všechny metody a properties ze třídy IpAddress a přidává k nim následující:

- **__init__(_bytes, mask):** Argument _bytes je předán konstruktoru rodičovské třídy. Argument mask je maska podsítě. Musí být nezáporná a nesmí větší než 32 pro IPv4 nebo 128 pro IPv6. Jinak bude vyvolána výjimka ValueError.
- **fromString(string):** Stejná statická metoda jako u třídy IpAddress. Liší se jenom formát textového řetězce string. Za adresu se doplní lomítko následované maskou. Například "192.0.2.1/24"nebo "2001:DB8::1/32".
- **getSupernet(family):** Statická metoda. Vytvoří objekt s adresou 0.0.0.0/0 nebo ::0/0 podle rodiny specifikované argumentem family.
- **netMask:** Property. Vrací masku podsítě.
- **netAddr:** Property. Spočítá adresu sítě a vrátí ji jako objekt IpAddress.
- **brdAddr:** Property. Spočítá adresu všesměrového vysílání (broadcast) a vrátí ji jako objekt IpAddress. Rodina IPv6 tyto adresy nepodporuje. Pro IPv6 adresu vrací None.

Operátor rovnosti je zděděn beze změny. Nebere v úvahu masku podsítě.

IpNetDetailed:

Třída rozšiřuje třídu IpNet o informaci o rozsahu platnosti adresy (scope) a informaci o tom zda adresa má v všesměrovou adresu. Obě informace využívá třída Interface popsaná v části 3.3.

Doplněné nebo přepsané metody a properties proti třídě IpNet jsou:

- **__init__(_bytes, mask[, scope, withBroadcast]):** První dva argumenty jsou přímo předány zděděnému konstruktoru.

  - **scope:** Rozsah platnosti adresy specifikovaný konstantami RT_SCOPE_*. výchozí hodnotou je RT_SCOPE_UNIVERSE - adresa platná všude.
  - **withBroadcast:** Očekává hodnoty True (výchozí hodnota) nebo False. Je to ekvivalent parametru brd + konzolového příkazu ip při přidávání IP adresy.

- **fromString(string[, scope, withBroadcast]):** Tato statická metoda se liší od stejné metody ze třídy IpNet pouze dvěma dalšími nepovinnými parametry.
- **withBroadcast:** Property pro četní i zápis. Má stejný význam jako parametr v konstruktoru.
- **scope:** Property která, vrací oblast platnosti adresy.

Operátor rovnosti je i zděděn beze změny takže se porovnává pouze samotná IP adresa.

3.3 Správa síťových rozhraní a jejich adres - třída Interface

Tuto úlohu zapouzdřuje třída Interface.

Kostruktor Interface() by se neměl volat přímo. K získání objektů, které reprezentují jednotlivá rozhraní slouží dvě statické metody a to:

- **get():** Vrátí slovník zařízení, kde klíče jsou jejich názvy (eth0, lo, ...).
- **getIndexesDict():** Vrátí slovník, kde klíče jsou čísla zařízení, tak jak je očíslovalo jádro.

Tyto objekty obsahují několik properties. Pokud není uvedeno jinak, lze je pouze číst.

- **name:** Vrací řetězec se jménem zařízení.
- **index:** Vrací číslo zařízení přidělené jádrem.
- **mac:** Objekt typu MacAddress, který nese fyzickou adresu rozhraní. Do této property je možné zapisovat a nastavit libovolnou fyzickou adresu.
- **lowerUp:** Vrací True pokud ovladač zařízení hlásí, že je zařízení aktivní na první vrstvě
- **flags:** Vrací číslo typu int, které představuje příznaky zařízení. Některé příznaky jsou popsány v části 2.9.1 na straně 12
- **ip:** Vrací seznam všech IP adres na tomto zařízení, jako seznam objektů IpAddress.
- **up:** Vrací logickou hodnotu True, pokud je zařízení administrativně povoleno. Tato property je zapisovatelná. up = True odpovídá příkazu ip link set dev ... up

Kromě name a index se všechny další properties při každém čtení znova načítají. Ukazují tedy vždy aktuální stav.

Dále jsou zde metody pro práci s IP adresami:

- **ipAdd():** Přidá IP adresu.
- **ipDel():** Smaže IP adresu.

Obě metody očekávají jako parametr objekt typu IpAddress.

Vypsat základní informace o rozhraních a jejich IP adresách je velmi jednoduché. Tak jak to ukazuje následující kód:

```
from PyNetlink import Interface, IpNetdetailed

for rozhrani in Interface.get().values():
    print(rozhrani, *rozhrani.ip, sep="\n")
```

Výpis 9: Informace o rozhraních a jejich IP adresách.

Další příklad předvádí jak přidat IP adresy 192.0.2.1/24 na rozhraní s indexem 1. Poté skript vypíše všechny IP adresy na tomto rozhraní a na závěr adresu smaže.

```
from PyNetlink import Interface, IpNetDetailed

ip = IpNetDetailed.fromString("192.0.2.1/24")
rozhrani = Interface.getIndexesDict()[1]

rozhrani.ipAdd(ip)
print("Na rozhrani '" + rozhrani.name + "' jsou IP adresy:")
print(*rozhrani.ip, sep="\n")
rozhrani.ipDel(ip)
```

Výpis 10: Manipulace s IP adresami pomocí třídy Interface.

3.4 Třída NextHop

Objekt této třídy zapouzdřuje informaci o tom, kam se má provoz přeposílat. Je využívána třídou Route, kde každé pravidlo může využívat více těchto cílů.

Standardní konstruktor vypadá následovně NextHop(ip=None, dev=None, weight=1). Význam parametrů konstruktoru:

- **ip:** Objekt typu IpAddress. Je to adresa kam se má provoz přeposílat.
- **dev:** Rozhraní přes které se bude provoz odesílat. Objekt typu Interface
- **weight:** číselné vyjádření váhy této cesty.

Je nutné minimálně vyplnit buď parametr ip a nebo dev, jinak konstruktor vyvolá výjimku AttributeError.

Po vytvoření se objekt stává neměnným. Jeho vlastnosti lze pouze číst pomocí následujících properties:

- **dev:** Vrací zařízení jako objekt typu Interface nebo popřípadě None, pokud nebylo nastaveno.

- **family:** Vrací rodinu adresy nebo None, pokud adresa nebyla nastavena.
- **ip:** Vrací IP adresu jako objekt typu IpAddress a nebo None.
- **weight:** Vždy vrátí číslo odpovídající váze cesty.

Třída dále podporuje konverzi na řetězec pomocí funkce str() Pythonu.

3.5 Třída Route

Tato třída zapouzdřuje jedno směrovací pravidlo, včetně jeho dalších parametrů. Konstruktor vypadá následovně:

Route(network, nexthops[, metric, scope, rtype, protocol, tos])

- **network:** Adresa sítě, pro kterou je toto pravidlo platné, typu IpNet.
- **nexthops:** Cíle, kam se bude provoz přeposílat. Vždy seznam s minimálně jedním objektem typu NextHop.
- **metric:** Metrika - číselné vyjádření délky cesty.
- **scope:** Oblast platnosti pravidla. Zde jsou očekávány stejné konstanty jako v poli rtm_scope struktury RtMsg popsaná v části 2.11.1 na straně 16. Výchozí hodnota RT_SCOPE_UNIVERSE - pravidlo je platné všude.
- **rtype:** Typ směrovacího pravidla. I zde jsou očekávány konstanty jako v poli rtm_type struktury RtMsg popsané v části 2.11.1. Výchozí hodnota RTN_UNICAST.
- **protocol:** Konstanta, označující jakým způsobem pravidlo vzniklo. Výchozí hodnota RTPROT_STATIC - přidáno administrátorem.
- **tos:** Číselné označení druhu služby. Výchozí hodnota 0 - všechny druhy.

Třída dále nabízí čtení těchto údajů pomocí stejnojmenných properties. Kromě nich zde exitují ještě tyto:

- **multipath:** Vrací True pokud je v seznamu nexthops více jak jeden objekt.
- **family:** Vrací rodinu pravidla na základě adresy v parametru network.

Dále pravidlo umožňuje konverzi na řetězec pomocí Pythonovské fukce str() a podporuje také funkci len(), která vrací počet objektů v seznamu nexthops.

3.6 Správa směrování - třída RoutingTable

Třída RoutingTable zapouzdřuje jednotlivé tabulky směrovacích pravidel.

Na rozdíl od třídy Interface, instance této třídy se vytvoří voláním běžného konstruktoru RoutingTable([table]). Parametr table je nepovinný a určuje číslo tabulky. Pokud se nezadá, počítá se se standardní tabulkou číslo 254. To je ta, která se vypíše zavoláním linuxového příkazu ip route.

Tato třída má pouze jedinou property routes, která vrací obsah tabulky jako seznam objektů typu Route. Tím, že tyto objekty umožňují konverzi na řetězec je možné vypsat základní informace obsahu směrovací tabulky pomocí dvou řádků kódu, tak jak to ukazuje výpis 11.

```
from Pynetlink import RoutingTable

print(*RoutingTable().routes, sep="\n")
```

Výpis 11: Jedoduchý výpis hlavní směrovací tabulky.

Vytvoření a vložení nového pravidla do tabulky už je o něco komplikovanější. Vytvořme například pravidlo, které říká že do sítě 192.0.2.0/24 se dostaneme přes 127.0.0.1. nejprve je potřeba pravidlo složit. K tomu jsou potřeba dvě zmíněné IP adresy s tím, že druhá se zabalí do objektu typu NextHop.

```
from PyNetlink import *

networkIp = IpNet.fromString("192.0.2.0/24")
nexthopIp = IpAddress.fromString("127.0.0.1")

nexthop = NextHop(ip=nexthopIp)

route = Route(networkIp, nexthop)
```

Výpis 12: Vytvoření směrovacího pravidla.

Pravidlo máme připraveno v proměnné route. Nezbývá než vytvořit objekt směrovací tabulky a pravidlo do ní přidat pomocí metody routeAdd()

```
table = RoutingTable()
table.routeAdd(route)
```

Výpis 13: Vložení směrovacího pravidla do tabulky.

Pokud bychom chtěli toto pravidlo odstranit, je to možné udělat velmi jednoduše, pomocí metody routeDel(), které se toto pravidlo předá.

```
table.routeDel(route)
```

Výpis 14: Odstranění směrovacího pravidla z tabulky.

Poslední nezmíněnou metodou třídy RoutingTable je metoda multiAdd(). Vznikla jako snaha o co největší zefektivnění hromadného vkládání pravidel. Proti vkládání pravidel po jednom, voláním routeAdd() v cyklu, je hromadné vkládání přibližně o třetinu času rychlejší.

3.7 Vnitřní struktura balíčku

Abych zjednodušil rozšiřování tohoto balíčku, obsahuje další vnořený balíček s názvem netlink. Tento balíček poskytuje modul constants, který obsahuje veškeré zde použité konstanty. Kromě něj balíček poskytuje ještě třídy Netlink a Message. Ty budou popsány dále.

3.7.1 Třída Message

Třída Message představuje mechanismus jak vytvořit strukturu požadavku, a nebo naopak dekódovat odpověď. Inspiroval jsem se filozofií kolekce namedtuple, která je standardní součástí Pythonu. Vytvořil jsem obdobnou kolekci, která umožňuje zanořování objektů do sebe. Vybavil jsem ji nástroji, které zjednodušují převod zprávy na formát bytes a zpět a také nástroji usnadňující ladění.

Práce je velmi podobná práci se zmiňovanou kolekcí namedtuple. Jak je patrné z příkladu níže nejprve se zavolá funkce Message(name, fmt), jejíž mají následující význam:

- **name:** Jméno nově vytvářených objektů.
- **fmt:** Řetězec reprezentující formát a názvy polí v hlavičce.

Řetězec formátu je vždy písmeno označující datový typ pole následované dvojtečkou a názvem tohoto pole. Jednotlivá pole se od sebe oddělují bílým znakem. Datový typ se specifikuje stejně jako u třídy Struct. Jsou uvedeny v tabulce 3 na straně 7. Prázdné pole s formátem x nemusí následovat název.

Při implementaci této třídy jsem musel zohlednit také fakt, že některé hlavičky obsahují pole s délkou dat a jiné ne. Pokud v hlavičce toto pole chybí, spadá pod hlavičku veškerý zbytek následujících dat, až do konce zprávy. Například RtMsg tak jak je patrné z tabulky 12 na straně 18. Naopak pokud hlavička pole s délkou má, spadá pod tuto hlavičku pouze část dat. Objekty třídy Message toto řeší jednoduše tak, že pole s délkou se musí jmenovat length. Na základě existence nebo neexistence tohoto pole funguje statická metoda fromBytes(). Pozná tak, kolik dat z předaného řetězce má přidat do atributu payload.

Pole length, pokud je ve struktuře obsaženo, je tedy vždy nutné inicializovat jakoukoliv hodnotou. Na správnou hodnotu bude upraveno v době převádění na bytes. Aktuální hodnotu délky vždy zjistí funkce len().

Volání Message() nám vrátí funkci, nebo přesněji funktor, což je volatelná třída, která se chová jako konstruktor objektů v daném formátu. Tvoření těchto objektů je poté velmi rychlé a pohodlné.

Zároveň s konstrukorem, vznikne i statická metoda fromBytes, pomocí níž je možné strukturu vytvořit i z řetězce bytes, který se předá do argumentu.

```
from PyNetlink.netlink import Message
from PyNetlink.netlink.constants import *

NlMsgHdr = Message('NlMsgHdr', 'I:length H:type H:flags I:seq I:pid')
```

```
nlmsghdr = NlMsgHdr(-1, RTM_GETROUTE, NLM_F_REQUEST | NLM_F_DUMP, 0, 0)
print(nlmsghdr)
```

Výpis 15: Vytvoření NlMsgHdr pomocí Message.

Výpis 15 ukazuje jak vytvořit objekt odpovídající hlavičce NlMsgHdr. Tento objekt je možno vypsat čitelně lidsky pomocí str(nlmsghdr) a nebo převést na bytes pomocí funkce bytes(NlMsgHdr). Dále tyto objekty zahrnují následující metody a porperties:

- **attributes:** Property, která vrací namedtuple s proměnnými reprezentující jednotlivá pole v hlavičce.
- **payload:** Property, která vrací data za hlavičkou ve formě seznamu. V tomto seznamu mohou být jak další objekty Message tak řetězce bytes vždy bez Případné výplně.
- **addPayload(*payloads):** Pomocí této metody je možné za hlavičku přidávat další data. Jak typu Message tak bytes. Případné zarovnání těchto dat za hlavičkou na délku dělitelnou čtyřmi se provede automaticky při konverzi na bytes. Do parametru je možné přidávat více dat najednou.
- **unpackPayload(cls):** Tato metoda vytvoří z bytes uložených v payload objekty třídy předané do parametru cls cls
- **printTree():** Tato metoda velmi užitečná při ladění aplikace. Vypíše strukturu vnořených objektů.

Nyní kód ve výpisu 15 rozšíříme tak aby představoval žádost o přidání výchozí brány. Aby tento příklad nic nerozbil vložíme ji do tabulky 100 a jako adresu brány zvolíme třeba 127.0.0.2. vytvoříme strukturu RtMsg. NlAttr je dostupná přímo v balíčku netlink. Následně složíme části do sebe a pomocí metody printTree() si necháme vzniklou zprávu zobrazit. Na závěr zprávu převedeme za bytes

```
from PyNetlink.netlink import NlAttr
from PyNetlink import IpAddress
RtMsg = Message('RtMsg', 'B:family␣B:dst_len␣B:src_len␣B:tos␣B:table␣B:protocol␣B:scope␣B:type␣I
    :flags')

ip = IpAddress.fromString("127.0.0.2")
rtmsg = RtMsg(ip.family, 0, 0, 0, 100, RTPROT_STATIC, RT_SCOPE_UNIVERSE,
    RTN_UNICAST, 0)
attr = NlAttr(-1, RTA_GATEWAY)

nlmsghdr.addPayload(rtmsg)
rtmsg.addPayload(attr)
attr.addPayload(bytes(ip))

nlmsghdr.printTree()
zprava = bytes(nlmsghdr)
```

Výpis 16: Vytvoření RtMsg pomocí Message.

Proměnná zprava nyní obsahuje řetězec bytes který představuje vytvořenou zprávu. Poslední část této ukázky předvede opačný postup. Nejprve vytvoří z bytes hlavičku NlMsgHdr a poté postupným voláním metody unpackPayload() i další hlavičky.

```
nlmsghdr = NlMsgHdr.fromBytes(zprava)
nlmsghdr.unpackPayload(RtMsg)
nlmsghdr.payload[0].unpackPayload(NlAttr)

nlmsghdr.printTree()
```

Výpis 17: Čtení bytes pomocí třídy Message.

Výstupy výpisů 16 i 17 by se měly lišit pouze v proměnných length. Skutečná velikost je do nich dosazena až při převodu na bytes. Pokud zaměníme pořadí posledních dvou řádků výpisu 16, budou výstupy naprosto stejné:

```
NlMsgHdr(length=36, type=24, flags=1537, seq=0, pid=0)
  RtMsg(family=2, dst_len=0, src_len=0, tos=0, table=100, protocol=4, scope=0, type=1, flags=0)
    NlAttr(length=8, type=5)
      Raw bytes: [b'\x7f\x00\x00\x02']
```

Výpis 18: Výstup výpisu 17.

3.7.2 Třída Netlink

Tato třída, jak její název napovídá, zapouzdřuje Netlink socket. Stará se o skládání a rozebírání standardní hlavičky NlMsgHdr a převádí případné chybové zprávy Netlinku na výjimky. Třída nijak neřeší souběhy.

Instance se vytvoří běžně konstruktorem Netlink().

Odeslání zprávy se provede pomocí metody send(hdrType, hdrFlags, *data). První dva parametry odpovídají polím hlavičky NlMsgHdr. Některé hodnoty, kterých mohou nabývat, jsou popsány v části 2 na straně 6.

- **hdrType:** Odpovídá poli nlmsg_type.
- **hdrFlags:** Odpovídá poli nlmsg_flags.

Za těmito parametry následuje jeden a nebo více zpráv ve formě objektů typu Message nebo obecně čehokoliv, co lze převést pomocí funkce bytes(). Pro každou z těchto zpráv bude sestavena hlavička NlMsgHdr a poté bude odeslána.

Metoda send() neumí sama vyřešit odeslání žádosti složené z více zpráv. Jednoduše proto, že jsem nenarazil na případ kdy by to bylo potřeba.

Příjem zpráv má na starosti metoda recvList() a nemá žádné parametry. Metoda přečte celou odpověď a to i když se skládá z více zpráv. Vrátí ji ve formě seznamu objektů typu NlMsgHdr. Pokud je přijímaná zpráva typu NLMSG_ERROR = 2, čili jde o chybovou zprávu viz. část 2.5 strana 9, bude vygenerována výjimka RuntimeError s řetězcem popisujícím chybu. Potvrzovací zprávy (Ack) jsou zamlčeny.

4 Závěr

Asi nejefektivnější způsob, jak zjistit jakým způsobem provést požadovanou akci, přes Netlink socket, je následující: Stažení zdrojových kódů utility ip a jejich přeložení s podporou ladících symbolů. Pomocí nástroje gdb krokovat provádění požadované akce touto utilitou. gdb umožňuje přehledné zobrazení odesílané struktury dat. Odesílaná data i případnou odpověď poté konfrontovat s RFC [3].

4.1 Zhodnocení dosažených výsledků

Tímto způsobem jsem vytvořil požadovaný balíček PyNetlink. Je napsán v Pythonu 3. Na verzi Pythonu nebyly kladeny žádné požadavky. Třetí verzi jsem zvolil jednoduše díky existenci výborné knihy Python 3 - Výukový kurz [1].

Balíček umožňuje provádění následujících operací:

- Získat seznam všech síťových rozhraní.
- Pro každé rozhraní umožňuje:
  - Číst jeho název, a informaci o stavu fyzické vrstvy.
  - Číst nebo změnit fyzickou adresu.
  - Číst nebo změnit administrativní stav zařízení (up/down).
  - Přidávat nebo odebírat IP adresy. Včetně informací o rozsahu platnosti (scope) a broadcast adresy.
- Modifikace směrovacích tabulek vkládáním nebo odebíráním směrovacích pravidel včetně pravidel pro více cílů a výchozích bran.

Obsahuje také sadu tříd, které zapouzdřují adresy a směrovací pravidla, aby se s nimi lépe a jednodušeji pracovalo. Uvnitř balíčku je vnořený balíček pojmenovaný netlink. Poskytuje třídy zapouzdřující komunikaci přes Netlink socket modul s konstantami.

Použití balíčku dokumentuje kapitola 3 této práce i včetně příkladů. Zdokumentován je i vnitřní balíček netlink, zejména proto že poskytuje nástroje, které zjednodušují další vývoj.

Kapitola 2 popisuje, jak provádět operace přímo přes Netlink socket. Taktéž zahrnuje doporučení, které standardní nástroje Pythonu k tomu s výhodou použít.

4.2 Možnosti dalšího rozšíření balíčku

V současném stavu by se balíček mohl stát součástí programů sloužících k nastavování síťových připojení. Myslím, že splňuje veškeré požadavky na nastavení pevné sítě.

Balíček je možné dále rozšiřovat prakticky tak, aby zahrnul veškeré možnosti, které konfigurace síťového subsystému přes Netlink socket nabízí. Vhodný směr dalšího vývoje závisí na způsobu využití balíčku:

- Pokud by byl balíček využit poskytovatelem internetového připojení pro konfiguraci zařízení, tento by určitě uvítal následujíc funkcionalitu:
  - Správu sousedů. Ekvivalent příkazu `ip neigh`. Staticky konfigurované záznamy v ARP tabulce znemožňují podvržení jiné fyzické adresy a odposlouchávání komunikace touto cestou. Tzv. ARP spoofing.
  - Správa pravidel kdy použít jakou směrovací tabulku. Ekvivalent příkazu `ip rule`.
  - Správa firewallu (iptables). I toto lze řešit přes Netlink socket.h
  - Možnost vytváření a mazání virtuálních síťových rozhraní IEEE 802.1Q (VLAN).
- Balíček by mohl být také využit jako součást směrovacího démona. Ovšem jedině za předpokladu, že by šlo o směrování menšího počtu sítí a nebo při vývoji tohoto démona. Například při praktickém ověření funkčnosti směrovacího algoritmu a nebo protokolu. Plnit směrovací tabulku pravidly pro veškeré IPv4 rozsahy celého internetu pomocí tohoto balíčku by trvalo příliš dlouho. Každopádně směrovací démon by určitě využil následující:
  - Informaci o změnách Adres na rozhraních ve formě událostí. Takto by démon mohl zjistit, že se na zařízení objevila nebo zmizela cesta do další podsítě. Navíc v nové podsíti možno, na základě této informace, začít vyhledávat další směrovače.
  - Informaci o tom že ve směrovací tabulce přibylo, nebo bylo smazáno směrovací pravidlo. Na základě těchto informací je možné například předávat dalším směrovačům pravidla, která vytváří jiný démon, popřípadě administrátor.

5 Přílohy

1. Disk CD
    - **BalaraskaPrace.pdf:** Tento text.
    - **PyNetlink:** Samotný balíček.
    - **InterfaceDemo.py:** Skript demonstrující možnosti třídy `Interface`.
    - **RoutingTableDemo.py:** Skript demonstrující možnosti třídy `RoutingTable`.

6 Reference

[1] SUMMERFIELD, Mark. *Python 3: výukový kurz*. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 584 s. ISBN 978-80-251-2737-7.

[2] DHANDAPANI, Gowri a Anupama SUNDARESAN. *Netlink Sockets - Overview* [online]. September 13, 1999 [cit. 2013-05-06]. Dostupné z: http://qos.ittc.ku.edu/netlink/netlink.pdf. The University of Kansas.

[3] RFC 3549 Linux Netlink as an IP Services Protocol http://www.rfc-editor.org/rfc/pdfrfc/rfc3549.txt.pdf

[4] struct — Interpret bytes as packed binary data. *Python v3.3.1 documentation* [online]. 6.5.2013 [cit. 2013-05-06]. Dostupné z: http://docs.python.org/3.3/library/struct.html